國立交通大學

資訊科學與工程研究所

碩士論文

個人特徵發現之手機應用程式預測
On the Feature Discovery for App Usage Prediction

研究生：李守峻

指導教授：彭文志 教授

中華民國一百零二年七月

個人特徵發現之手機應用程式預測

On the Feature Discovery for App Usage Prediction

研究生 ： 李守峻　　Student : Shou-Chung Li

指導教授 ： 彭文志　Advisor : Wen-Chih Peng

國立交通大學

資訊科學與工程研究所

碩士論文

A Thesis
Submitted to Institute Of Computer Science and Engineering
College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in

Computer Science

July 2013

Hsinchu, Taiwan, Republic of China

中華民國一百零二年月七月

個人特徵發現之手機應用程式預測

學生 ： 李守峻　　　　　　　　　　　　指導教授 ： 彭文志

國立交通大學資訊科學與工程研究所

摘要

隨著越來越多的手機應用軟件的開發，它們正密切地融入人們的日常生活中。在本論文中，我們發展了一個基於當前設備狀態以預測智能手機上最有可能被使用的應用軟件的框架。這樣的應用程序使用的預測框架對於加速應用程式發動，智能手機的用戶體驗，和電源管理都是不可或缺的條件.透過真實的應用程序使用日誌數據的分析，我們發現兩種特徵數值：從內置的感測器所測得讀數，稱為顯性特徵（EF），以及從應用程序使用關係轉換的數值，稱為隱性特徵。IF 特徵數值是從建構應用程序使用轉變的應用程序使用圖（簡稱ＡＵＧ）的模型而推衍。鑒於ＡＵＧ的圖形，我們能夠發現應用程序之間的關係。由於用戶可能有不同的在智能手機的使用行為，我們進一步提出一個個人化的特徵選擇演算法。從訓練數據裡，我們探索最低描述長度（MDL），並選擇需要較少位元數來形容訓練數據資料的特徵。個人特徵選擇演算法，可以成功地減少日誌的大小和預測所花費的時間。最後，我們採用 KNN 分類模型來預測應用程序的使用。需要注意的是當我們使用 k 近鄰分類器時，我們只需要保留通過個人化特徵選擇演

算法的特徵，這不僅可以降低預測的時間又能避免多維度所帶來的缺點。最後，我們做了在真實行動應用程式的數據集的一個完整綜合的實驗研究。得到的結果表示，我們提出的框架是有效的並顯示其在應用程式使用上的預測的能力。

On the Feature Discovery for App Usage Prediction

Student : Shou-Chung Li　　　　　　Advisor : Dr. Wen-Chih Peng

Institute Of Computer Science and Engineering,
National Chiao Tung University

ABSTRACT

With the increasing number of mobile Apps developed, they are now closely integrated into daily life. In this paper, we develop a framework to predict mobile Apps that are most likely to be used regarding the current device status of a smartphone. Such an Apps usage prediction framework is a crucial prerequisite for fast App launching, intelligent user experience, and power management of smartphones. By analyzing real App usage log data, we discover two kinds of features: The Explicit Feature (EF) from sensing readings of built-in sensors, and the Implicit Feature (IF) from App usage relations. The IF feature is derived by constructing the proposed App Usage Graph (abbreviated as AUG) that models App usage transitions. In light of AUG, we are able to discover usage relations among Apps. Since users may have different usage behaviors on their smartphones, we further propose one personalized feature selection algorithm. We explore minimum description length (MDL) from the training data and select those features which need less length to describe the training data. The personalized feature selection can successfully reduce the log size and the prediction time. Finally, we adopt the kNN classification model to predict Apps usage. Note that through the features selected by the proposed personalized feature selection algorithm, we only need to keep these features, which in turn reduce the prediction time and avoid the curse of dimensionality when using the kNN classifier. We conduct a comprehensive experimental study based on a real mobile App usage dataset. The results demonstrate the effectiveness of the proposed framework and show the predictive capability for App usage prediction.

致謝

在研究所的兩年裡我學到很多專業的知識與報告技巧,也學會了如何面對、思考、解決研究上的問題，即使這兩年來在就讀碩士班的過程遇到許多困難與挫折，但也因此從中獲益不少。這一路走來有許多人要感謝，使得我可以在這碩士生涯的最後寫下這篇致謝文。

首先誠摯的感謝指導教授彭文志博士，老師悉心的教導使我得以領悟手機應用程式使用行為辨識領域的深奧，不時的討論並指點我正確的方向，使我在這些年中獲益匪淺。老師對學問的嚴謹更是我輩學習的典範。

本論文的完成亦得非常感謝學長 Dimension 的大力協助。因為有您的細心的教導及熱心的幫忙，使得本論文能夠更完整而嚴謹，讓我得以順利畢業，這份恩情我會永久銘記在心，並期許我自己有一天可以像學長一樣成為一位值得信賴的領導者。

兩年裡的日子，實驗室裡共同的生活點滴，學術上的討論、言不及義的閒扯、實驗室的出遊、霸佔研討室玩桌遊或是看電影的蠻橫、趕投稿的革命情感，感謝眾位學長姐、同學、學弟妹的共同砥礪，你/妳們的陪伴讓兩年的研究生活變得絢麗多彩。

感謝 Barry 學長、拍拍、許雅婷、01 學姐們不厭其煩的指出我研究中的缺失，且總能在我迷惘時為我解惑，也感謝徐宗豪、林詠翔、周凡凱、陳建志同學的幫忙，恭喜我們順利走過這兩年。實驗室的彭湞湘學弟、孫星星、溫郁婷、許宛婷、阮曉雯學妹們當然也不能忘記，你/妳們在口試當天的協助我銘感在心。

另外，我要特別感謝這兩年來家人的包容，爸爸、媽媽、姊姊，你們是我最大的精神支柱，在我忙於碩士論文研究之餘，給予我很多的支持與鼓勵，不管是精神上或是物質上的支助，讓我得以無憂地專注做好研究。

最後，謹以此文獻給我摯愛的大家。

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the increasing number of smartphones, mobile applications (Apps) have been developed rapidly to satisfy users' needs [28, 4, 23, 26]. Users can easily download and install Apps on their smartphones to facilitate their daily lives. For example, users use their smartphones for Web browsing, shopping and socializing [17, 3]. By analyzing the collected real Apps usage log data, the average number of Apps in a user's smartphone is around 56. For some users, the number of Apps is up to 150. As many Apps are installed on a smartphone, users need to spend more time swiping screens and finding the Apps they want to use. From our observation, each user has on average 40 launches per day. In addition, the launch delay of Apps becomes longer as their functionality becomes more complicated. In [27], the authors investigated the launch delay of Apps. Even simple Apps (e.g., weather report) need 10 seconds, while complicated Apps (e.g., games) need more than 20 seconds to reach a playable state. Although some Apps could load stale content first and fetch new data simultaneously, they still need several seconds to complete loading.

To ease the inconvenience of searching for Apps [14, 24] and to reduce the delay in launching Apps [27], one possible way is to predict which Apps will be used before the user actually needs them. Although both the iOS and Android systems list the most recently used (MRU) Apps to help users relaunch Apps, this method only works for those Apps which would be immediately relaunched within a short period. Another common method is to predict the most frequently used (MFU) Apps. However, when a user has a lot of frequently used Apps, the MFU method has very poor accuracy. In our experiments, these two methods are the baseline methods for comparison.

Recently, some research works have addressed the Apps usage prediction problems [27, 14, 24]. In [14], a temporal profile is built to represent the usage history of an App. The temporal profile records the usage time and usage period of the App. Then, when a query time is given, the usage probability of each App could be calculated through comparing the difference between the temporal profile and the query time. However, since they only consider the periodicity feature of Apps, some Apps with no significant periods cannot be predicted by their temporal profiles. In [27], the authors adopted three features to predict Apps usage: time, location, and used Apps. Based on those three features, they designed and built a system to remedy slow App launches. However, they always use these three features to predict different users' usage, which is impractical as users could have different usage behavior. For example, the location information could be less useful for those users who have lower mobility. We claim that the features which are able to accurately predict Apps usage are different for different users and different Apps. The authors in [24] collected 37 features from accelerometer, Wi-Fi signal strength, battery level, etc., and proposed a Naive Bayes classification method to predict Apps usage. However, a Naive Bayes classification method needs sufficient training data to calculate the conditional probability, which does not always hold. Therefore, the system would fail to predict Apps if there are not exactly the same instances existing in the training dataset. In addition, they still apply all the same features to each user, instead of selecting personalized features for different users with different usage behaviors.

In this chapter, we adopt the concept of minimum description length (MDL) to select personalized features for different users and propose a kNN-based App Prediction framework, called KAP, to predict Apps usage. Once we distinguish the useful and useless features, only the useful features need to be collected. Therefore, the size of the log data could be reduced. The overall framework is shown in Figure 1.1. KAP investigates features from both explicit and implicit aspects. The explicit feature is a set of sensor readings from built-in hardware

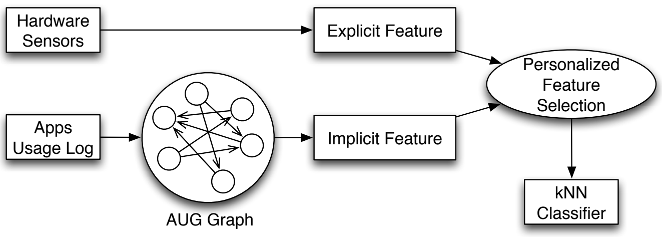


Figure 1.1: Overview of kNN-based App Prediction framework.

sensors, such as GPS, time, accelerometers, etc. On the other hand, the implicit feature is referred to as the correlations of Apps usage. To capture these correlations, the implicit feature is represented as the transition probability among Apps.

For the explicit feature, we focus on three types of hardware sensors: 1) device sensors, such as free space, free ram, and battery level, 2) environmental sensors, such as time, GSM signal, and Wi-Fi signal, and 3) personal sensors: acceleration, speed, heading, and location. We claim that the usage of different Apps is related to different types of sensors. Obviously, the advantages of selecting sensors for the explicit feature is that it reduces the effect of noisy data and also saves power and storage consumption for logging data and performing the prediction.

For the implicit feature, we calculate the transition probability for each App. However, the previous works [27, 24] only take the usage order into account, and not the time duration between Apps. We claim that the length between Apps usage means different things. For example, users may take pictures via a camera App and upload those pictures to Facebook. However, some users may upload pictures immediately, while others would upload them when they have a Wi-Fi connection. Therefore, the time duration between camera and Facebook use depends on different users and different usage behaviors. To model the usage relation among Apps, an Apps Usage Graph (AUG), which is a weighted directed graph, is proposed. The weight on each edge is formulated as an exponential distribution to describe the historical usage durations. Based on AUG, the implicit feature of each training instance is derived by traversing the AUG. Consequently, the implicit feature of each testing case is derived by an iterative refinement process.

With both explicit and implicit features, KAP adopts a kNN classification model to predict Apps usage which is represented as class labels. In the experimental study, the proposed KAP framework outperforms both baseline methods and achieves accuracy of 95%. We also show that the personalized sensor selection for the explicit feature is efficient and effective. In addition, the implicit feature is useful for improving the prediction accuracy of KAP.

The major contributions of this research work are summarized as follows.

- We address the problem of Apps usage prediction by discovering different feature sets to fulfill different users' Apps usage behavior, and propose the concept of explicit and implicit features for Apps usage prediction.

- We estimate the distribution of the transition probability among Apps and design an Apps Usage Graph (AUG) to model both Apps usage order and transition intervals. Two algorithms are proposed to extract the implicit features from the AUG graph for training and testing purposes respectively.

- We propose a personalized feature selection algorithm in which one could explore MDL to determine a personalized set of features while still guaranteeing the accuracy of the predictions.

- A comprehensive performance evaluation is conducted on real datasets, and our proposed framework outperforms the state-of-the-art methods [24].

The rest of this chapter is organized as follows. Chapter 2 investigates the related works which discuss the conventional prediction problem and Apps usage prediction. Chapter 3 introduces the explicit and implicit features. Chapter 4 presents the mechanism of personalized feature selection. Chapter **??** conducts extensive and comprehensive experiments. Chapter 4 depicts the classification of data instances. Finally, this paper is concluded with Chapter 7.

Chapter 2

Related Works

To the best of our knowledge, the prediction problem of Apps usage in this chapter is quite different from the conventional works. We focus on not only analysing usage history to model users' behavior, but on personalizing varied types of features including hardware and software sensors attached to smartphones. The proposed algorithm selects different features for different users to satisfy their usage behavior. Although there have been many research works solving the prediction problem in different domains, such as music items or playlist prediction [2], dynamic preference prediction [15, 12], location prediction [13, 22, 19], social links prediction [5, 16], and so on, the prediction methods are only based on analysing the usage history. In [11], the author selected features from multiple data streams, but the goal is to solve the communication problem in a distributed system.

Currently, only a few studies discuss mobile Apps usage prediction. Although the authors in [18] adopted location and time information to improve the accuracy of Apps usage prediction, the total number of Apps is only 15. Concurrently, in [8], the authors stated that the prediction accuracy could achieve 98.9%, but they still only focus on predicting 9 Apps from a set of 15. In [27], the authors solved the prediction problem through multiple features from 1) location, 2) temporal burst, and 3) trigger/follower relation. However, they did not analyze the importance of each feature. Therefore, for different users, they always use the same three features to predict their Apps usage. In [24], the authors investigated all possible sensors attached to a smartphone and adopted a Naive Bayes classification to predict the Apps usage. However, collecting all possible sensors is inefficient and impractical. Moreover, the useful sensors for different users could vary according to users' usage behavior. We claim that for different users, we need to use different sets of features to predict their usage. In this chapter, we collect only the subset of all features which are personalized for different users.

This chapter is the first research work which discusses how to select suitable sensors and features for different users to predict their Apps usage. Through the personalized feature selection, we could perform more accurate predictions for varied types of usage bahavior, reduce the dimensionality of the feature space, and further save energy and storage consumption. In addition, the proposed KAP framework derives the implicit feature by modelling the usage transition among Apps.

Chapter 3

Explicit and Implicit Features

In this chapter, we separate the features into two main categories: the explicit feature and the implicit feature. The explicit feature represents the sensor readings which are explicitly readable and observable. The implicit feature is the Apps usage relations.

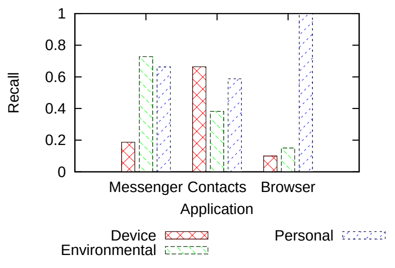
3.1 Explicit Feature Collection

Table 3.1 shows the hardware sensors we use for the explicit feature. As different models of smartphones could have different sets of hardware sensors, we only list the most common ones whose readings are easy to record. It is totally free to add or remove any hardware sensors here.

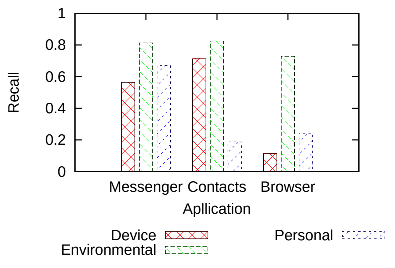
To show the prediction ability of different types of mobile sensors, we randomly select two users from the collected dataset and perform kNN classification via the three types of sensors respectively to predict their Apps usage. Figure 3.1 shows the prediction recall of "Messenger", "Contacts", and "Browser" for the two users. As can be seen in Figure 3.1, personal sensors would be a good explicit feature for predicting $user_1$'s Apps usage, while environmental sensors are good for $user_2$. The reason is that $user_2$ probably needs a Wi-Fi signal to access the Internet.

Table 3.1: Hardware sensors for the explicit feature.

Sensors	Contextual Information
Location	Longitude Latitude Altitude Location Cluster
Time	Hour of day Day of week
Battery	Battery Level Charging status
Accelerometer	Avg. and std. dev. of {x, y, z} Acceleration changes speed Heading
Wi-Fi Signal	Received signal
GSM Signal	Signal Strength
System	Free space of each drive Free RAM



(a) $User_1$



(b) $User_2$

Figure 3.1: Varied recalls of predicting Apps usage via different types of sensors for different users.

3.2 Implicit Feature Extraction

The implicit feature formulates the usage transitions among Apps in a usage session. As mentioned in [27], users use a series of Apps, called a usage session, to complete a specific task. For example, one user could use "Maps" when travelling to a sightseeing spot, then use camera to take photos, and upload those photos to Facebook. Thus, the series of using "Maps", "Camera" and "Facebook" is called a usage session, denoted as "Map"$\xrightarrow{\delta_1}$"Camera"$\xrightarrow{\delta_2}$"Facebook", where δ_1 and δ_2 represent the transition intervals.

The implicit feature of "Facebook" in this usage session is thus $< p_{MF}(\delta_1), p_{CF}(\delta_1 + \delta_2), p_{FF}(\infty) >$, where $p_{MF}(\cdot)$, $p_{CF}(\cdot)$, and $p_{FF}(\cdot)$ are probability models which represent the probability of using "Maps", "Camera" and "Facebook" respectively before using "Facebook" with the transition interval as the random variable. Note that because there is no "Facebook" to "Facebook" in this usage session, the transition interval is thus set to ∞ and then the probability would be 0.

The probability model could be estimated from a user's historical usage trace. In this section, we introduce an Apps Usage Graph (AUG) which models the transition probability among Apps for a single user. For training purposes, the implicit features for the training usage sessions are derived by traversing the AUG. However, for testing purposes, since we do not know which is the App to be invoked, the derivation of the implicit feature for the training usage session cannot be utilized directly. Therefore, an iterative refinement algorithm is proposed to estimate both the next App and its implicit feature simultaneously. The following paragraphs will illustrate the details of the AUG construction and the implicit feature derivation for both the training and testing usage sessions.

3.2.1 Apps Usage Graph (AUG)

For each user, we construct an Apps Usage Graph (AUG) to describe the transition probability among Apps. An AUG is a directed graph where each node is an App, the direction of an edge between two nodes represents the usage order, and the weight on each edge is a

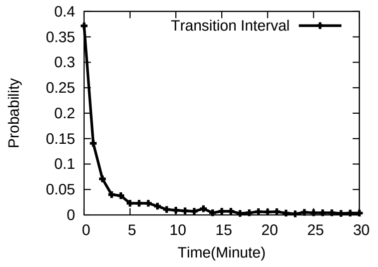


Figure 3.2: The PDF of the duration of two consecutive App launches.

probability distribution of the interval between two Apps. Since two consecutive launches could be viewed as a Poisson arrival process, we can formulate the intervals between two launches as an exponential distribution. For example, Figure 3.2 shows the probability density function (PDF) of two consecutive launches which exactly fulfils the exponential distribution where most transitions (e.g., 0.45%) are within 1 minute.

Here, Equation 3.1 formulates the exponential density function of the launch interval being in $[x, x+1)$. The parameter $\alpha = p(\hat{0})$ is derived by assigning $x = 0$ in Equation 3.1, and could be calculated by $p(0)$, the real probability derived from the training data. Then, β is solved by minimizing the difference between the estimated probability $p(\hat{i})$ and the real probability $p(i)$ as shown in Equation 3.2 for every interval i.

Empirically, we do not need to fit every interval when obtaining the exponential model. For example, in Figure 3.2, only the first 5 intervals already cover more than 75% of the training data. Therefore, we can iteratively add one interval until the data coverage reaches a given threshold. We will discuss the impact of the data coverage threshold in the experiments section.

$$p(\hat{x}) = \alpha \exp^{-\beta x} \tag{3.1}$$

$$\begin{aligned} \beta &= \underset{\beta}{\operatorname{argmin}} \sum_i |p(\hat{i}) - p(i)| \\ &= \underset{\beta}{\operatorname{argmin}} \sum_i |p(0) \exp^{-\beta i} - p(i)| \end{aligned} \tag{3.2}$$

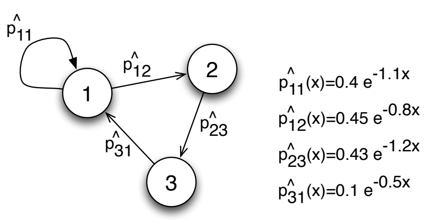

$\hat{p_{11}}$
$\hat{p_{12}}$
$\hat{p_{23}}$
$\hat{p_{31}}$
1
2
3
$\hat{p_{11}}(x)=0.4\ e^{-1.1x}$
$\hat{p_{12}}(x)=0.45\ e^{-0.8x}$
$\hat{p_{23}}(x)=0.43\ e^{-1.2x}$
$\hat{p_{31}}(x)=0.1\ e^{-0.5x}$

Figure 3.3: An example of the Apps Usage Graph (AUG).

For example, Figure 3.3 shows an AUG with three Apps. From Figure 3.3, the probability of two consecutive usages of App_1 with an interval of 0.3 minutes (i.e., $App_1 \xrightarrow{0.3} App_1$) is 0.4, and $App_1 \xrightarrow{1.5} App_2$ is 0.2. Although AUG only takes two consecutive Apps into account, such as p_{12} and p_{23}, the probability of p_{13}, could be calculated by $p_{12} \times p_{23}$.

3.2.2 Implicit Features for Training

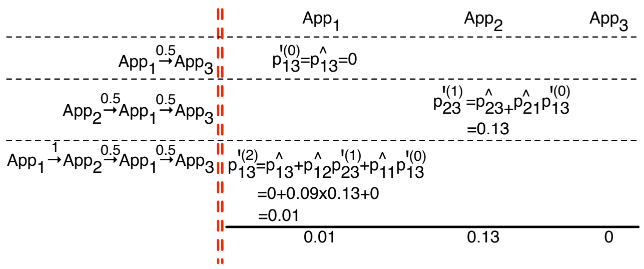
For each training case, the implicit features are derived by looking up the AUG. Suppose the currently used App (i.e., class label) is App_t, the implicit feature is thus, $< p'_{1t}, p'_{2t}, ..., p'_{nt} >$, where p'_{it} represents the probability of transiting from App_i to any random Apps and then to App_t. The probability of $p'^{(s)}_{it}$ is defined as in Equation 3.3 which is the summation of every probability from App_i to App_t. Note that we use a superscript, s, to indicate how many Apps are between App_i and App_t, and App_{m_k} is the k-th App after App_i. Once we derive the implicit feature in a reverse time order, the sub-problem of estimating $p'^{(s-k)}_{m_k,t}$ is already solved. The calculation of the implicit feature for App_i stops when the transition probability falls below a given threshold, min_{tp}. In our collected dataset, the transition probability falls to 0.1% when we look backward to more than 5 Apps, which is the default parameter for min_{tp}. Algorithm 1 depicts the derivation of the implicit feature for a training case with App_t as its class label.

$$p'^{(s)}_{it} = \hat{p_{it}} + \sum_k p_{i,\hat{m_k}} \times p'^{(s-k)}_{m_k,t} \tag{3.3}$$

For example, suppose we have an AUG as shown in Figure 3.3 and a usage trace as $\cdots \to App_1 \xrightarrow{1} App_2 \xrightarrow{0.5} App_1 \xrightarrow{0.5} App_3 \to \ldots$. Figure 3.4 shows the process of obtaining the implicit feature of App_3. We first estimate $p'^{(0)}_{13}$ from $App_1 \xrightarrow{0.5} App_3$, then $p'^{(1)}_{23}$ from

Algorithm 1: Deriving the implicit feature of App_t for training.

```
Input: $App_t$: a training App
Output: $IF_t$: the implicit feature of $App_t$

foreach $App_i$ prior than $App_t$ do
    $IF_t[i] \leftarrow IF_t[i] + p_{it}(\hat{\delta}_{it})$ ;
    foreach $App_m$ between $App_i$ and $App_t$ do
        $IF_t[i] \leftarrow IF_t[i] + p_{im}(\hat{\delta}_{jm}) \times IF_m[t]$ ;
    end
end
return $IF_t$
```



App$_1$
App$_2$
App$_3$
App$_1 \xrightarrow{0.5}$ App$_3$
$p'^{(0)}_{13} = \hat{p}_{13} = 0$
App$_2 \xrightarrow{0.5}$ App$_1 \xrightarrow{0.5}$ App$_3$
$p'^{(1)}_{23} = \hat{p}_{23} + \hat{p}_{21} p'^{(0)}_{13}$
$= 0.13$
App$_1 \xrightarrow{1}$ App$_2 \xrightarrow{0.5}$ App$_1 \xrightarrow{0.5}$ App$_3$
$p'^{(2)}_{13} = \hat{p}_{13} + \hat{p}_{12} p'^{(1)}_{23} + \hat{p}_{11} p'^{(0)}_{13}$
$= 0 + 0.09 \times 0.13 + 0$
$= 0.01$
0.01
0.13
0

Figure 3.4: Steps of obtaining the implicit feature of App_3 in the training case, $\cdots \rightarrow App_1 \xrightarrow{1} App_2 \xrightarrow{0.5} App_1 \xrightarrow{0.5} App_3$.

$App_2 \xrightarrow{0.5} App_1 \xrightarrow{0.5} App_3$, and finally update $p'^{(2)}_{13}$ from $App1 \xrightarrow{1} App_2 \xrightarrow{0.5} App_1 \xrightarrow{0.5} App_3$. Note that $p'^{(0)}_{13}$ is reused for calculating $p'^{(1)}_{23}$, and $p'^{(1)}_{23}$ and $p'^{(0)}_{13}$ are reused for updating $p'^{(2)}_{13}$. The implicit feature of App_3 is $< 0.01, 0.13, 0 >$.

3.2.3 Implicit Features for Testing

Since the App to be predicted for current invocation, App_t, is unknown for testing, the derivation process of implicit features for training does not work. We propose an iterative refinement algorithm to estimate both App_t and its implicit feature, IF_t, for testing. Suppose θ_i is the probability of $App_t = App_i$, the implicit feature IF_t is calculated as in Equation 3.4 which is a linear combination of the IF of each App_i. In addition, $M = [IF_1^T, IF_2^T, \ldots]$ represents the transition matrix among Apps, where IF_1^T, IF_2^T, $\ldots$ are column vectors. Then, the value of θ_i could be updated by Equation 3.5, which is the probability of staying in App_i after one-step walking along the transition matrix M. We keep updating θ_i and IF_t iteratively, until App_t

is fixed to one specific App. In our experiments, the iterative refinement process converges in about 3 iterations. Algorithm 2 depicts the derivation of the implicit feature for testing.

$$IF_t = \sum_{App_i} \theta_i \times IF_i \tag{3.4}$$

$$\theta_i = \sum_{App_m} IF_t[m] \times M[m][i] \tag{3.5}$$

Algorithm 2: Deriving the implicit feature for testing.

```
Input: $t$: a testing case
Output: $IF_t$: the implicit feature at $t$
while $iter < threshold$ do
    foreach $\theta_j$ do
        $IF_t \leftarrow IF_t + \theta_i \times IF_i$ ;
    end
    foreach $App_i$ prior than time $t$ do
        $\theta_i \leftarrow \theta_i + IF_t[m] \times M[m][i]$ ;
        Normalize $\theta_i$ ;
    end
    $iter \leftarrow iter + 1$ ;
end
return $IF_t$
```

For example, suppose the testing case is $\cdots \rightarrow App_1 \xrightarrow{1} App_2 \xrightarrow{0.5} App_1 \xrightarrow{0.5} App_t$. First, we initialize θ_i as $< 1/3, 1/3, 1/3 >$, which gives equal probability to each App, and the transition matrix $M = \begin{bmatrix} 0.49 & 0.6 & 0.01 \\ 0 & 0 & 0.13 \\ 0 & 0 & 0 \end{bmatrix}$, which is derived by calculating the IF of each App shown in Equation 3.3. Note that the last row is all zero because there is no App_3 transiting to any other Apps. Then, the implicit feature is $< 0.37, 0.04, 0 >$ in the first iteration. Next, θ_i is updated to $< 0.18, 0.22, 0.01 >$, and normalized as $< 0.44, 0.54, 0.02 >$ according to one-step walk in M with the calculated implicit feature as the prior probability. Then, we can obtain the implicit feature as $< 0.53, 0.01, 0 >$ in the second iteration.

Chapter 4

Personalized Feature Selection

The goal of the personalized feature selection is to use as fewer features as possible to guarantee an acceptable accuracy. Due to the energy and storage consumption of collecting sensors readings and Apps transition relations, we should select useful features for different users in advance. Furthermore, through the personalized feature selection, we could avoid the curse of dimensionality on performing the kNN. We first apply the personalized feature selection on the training data, and then only the selected features are required to be collected in the future.

Here, we propose a greedy algorithm to select the best feature iteratively. We adopt the concept of Minimum Description Length (MDL) [20, 21] to evaluate the goodness of the features. For different features, we can have varied projections of the training data. We claim that if a feature needs fewer bits to describe its data distribution, it is good for predicting the data. Therefore, in each iteration, the feature with the minimum description length is selected. Then, those data points which are correctly predicted are logically eliminated from the training data, and the next feature is selected by the same process repeatedly. We define the description length of the hypothesis, which is shown in Equation 4.1, as the length of representing the training data. $NG(App_i)$ is the number of groups of App_i. The description length of Data given the hypothesis is the total number of miss-classified data which is formulated as in Equation 4.2.

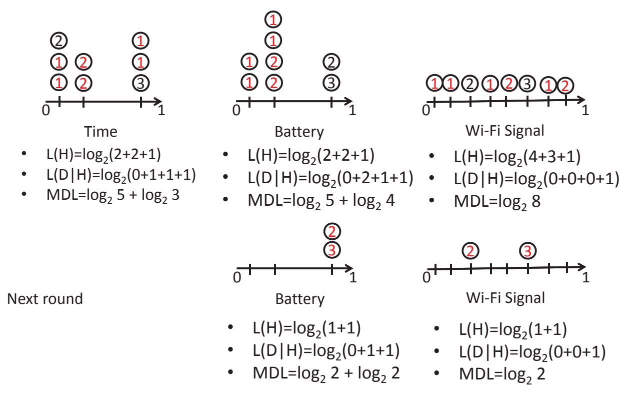


Figure 4.1: An example of feature selection where the red data points are correctly predicted.

$$L(H) = \sum_i \log_2 NG(App_i) \tag{4.1}$$

$$L(D|H) = \sum_i \log_2(missClassified(App_i) + 1) \tag{4.2}$$

For example, given 8 data points in the training data and three features as shown in Figure 4.1. In the first round, Time is the feature with minimum description length. Those data points marked as red are correctly predicted and will be removed. Therefore, in the second round, only two data points are left, and the feature of Wi-Fi signal is selected due to its minimum description length.

The selection process stops when a percentage of ρ of the training data is covered. We also discuss the impact of ρ in the experimental section. Note that the number of features affects the energy and storage consumption and is set according to the capability of the smartphones. Algorithm 3 depicts the process of personalized feature selection. After the selection, only the readings of the sensors which are selected will be collected as the explicit feature in the future. In addition, only the selected Apps will be used to construct AUG.

Algorithm 3: Personalized feature selection.

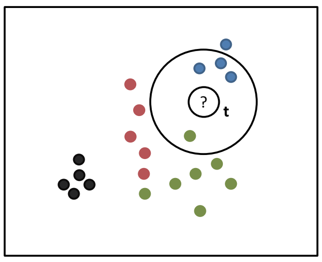
```
Input: $D_z$: the training data
Output: $PF$: the personalized features

Let $N_z \leftarrow |D_z|$ ;
while $|D_z| < \rho N_z$ do
    foreach feature $f$ do
        Calculate $DL_f$: description length for feature $f$ ;
    end
    $PF \leftarrow PF \cup \{\operatorname*{argmax}_f DL_f\}$ ;
    Let $D_a$ be the set of accurately predicted data points ;
    $D_z \leftarrow D_z - D_a$ ;
end
return $PF$
```

Chapter 5

Nearest-Neighbor Classifier

Nearest-neighbor classification is part of a more general technique known as instance-based learning, which uses specific training instances to make predictions without having to maintain a model derived from data. Instance-based learning algorithms requires a proximity measure to determine the similarity or distance between instances and a classification functions that returns the predicted class of test instance based its proximity to other instances.

A nearest-neighbor classifier represents each App usage trace as a data point in a n-dimensional space, where n is the number of features. Given a test point t, we compute its proximity to the rest of data points in the training set, using the proximity measures described in Equation(5.1). The k-nearest neighbors of a given test point t refer to the k points that are closest to t.



App	Recent Used Time
● Angry Birds	2013/7/26 11:30 PM
● Calendar	2013/7/26 08:30 AM
● Gmail	2013/7/26 11:00 PM
● Alarm	2013/7/26 01:10 AM
? Unknown	2013/7/26 11:55 PM

Figure 5.1: An example of nearest neighors of an instance

5.1 Weighted Voting

Once the nearest-neighbor list is obtained, the test point is classified based on the majority class of its nearest neighbors:

$$f(v) = \sum_{(x_i, App_i) \in D_t} w_i \times I(v = App_i) \times r_v \qquad (5.1)$$

where v is a class label, App_i is the class label for one of nearest neighbors, $I(\cdot)$ is an indicator function that returns 1 if its argument is true and 0 other wise, w_i is a distance factor to reduce the impact of k : $w_i = \frac{1}{d(\mathbf{x'}, x_i)^2}$, and r_i is a recency factor to reflect the importance of reusing Apps in a short period : $r_v = \frac{1}{2}^{\frac{t_{now} - t_v}{\lambda}}$.

For example, Figure 5.1 illustrates the 4-nearest neighbors of a test point located at the center of the circle and the recently-used time table of Apps. In the case where the neighborhood contains three Angry Bird and one Gmail usages and the distances between the test point and neighbors are 1. Therefore the test point is assigned to the Angry Bird by the voting scheme formed in Equation(5.1).

5.2 Nearest Neighbor Classification Algorithm

A brief schema of the nearest-neighbor classification method is given in Algorithm **??**. The algorithm computes the distance between the test point $t = (\mathbf{x'}, App')$ and all the training data $(\mathbf{x}, App) \in D_z$ to determine its nearest-neighbor list, D_t where $\mathbf{x'}$ is the feature vector of t, App' is the ground-true, $\mathbf{x}$ is a feature vector of a training point, and App is a class label of a training point. Such computation can be costly if the number of training examples is large. However, efficient indexing techniques can be found in [10, 9, 25] to reduce the amount of computation costs to search the nearest neighbors of a test case.

Algorithm 4: K-nearest Neighbor Classification Algorithm

```
Input: $t$: a test point
Output: $L$: a top-$k$ prediction list

Compute $d(\mathbf{x'}, \mathbf{x})$, the distance between $t$ and every example, $(\mathbf{x}, App) \in D_z$.
Select $D_t \subseteq D_z$, the set of k closest training data to $t$
$L \leftarrow$ the top-k App list ranked by Equation(5.1)
return $L$
```

Chapter 6

Experimental study

In this section, we conduct a comprehensive set of experiments to compare the performance of the proposed KAP framework with other existing methods including 1) most frequently used (MFU) method, 2) most recently used (MRU) method which is the built-in prediction method in most mobile OS, such as Android and iOS, 3) SVM, 4) App Naive Bayes [24], 5) Decision Tree, and 6) AdaBoost. In the following, we first discuss the collected dataset, then introduce the metrics employed to evaluate the performance, and finally deliver the experimental results.

6.1 Dataset Description

In this paper, we use a real world dataset collected by a mobile phone company which installed a monitoring program on every volunteer's smartphone. In this dataset, we have totally 50 volunteers including college students and faculty from June 2010 to January 2011. For each user, we separate the dataset into three parts, where each part consists of three months, and we use the first two months as training data, and the last one month as testing data. Totally, there are more than 300 different Apps installed on their smartphones, and the average number of Apps on one smartphone is 56.

6.2 Performance Metrics

In this paper, we use two performance metrics: 1) average recall and 2) nDCG [6] score.

Average Recall: Since there is only one App being launched in each testing case, recall score is thus adopted as one performance metric which evaluates whether the used App is in the prediction list. The recall score of one user is defined as $\sum_{c_i \in C} \frac{I(App_{c_i}, L_{c_i})}{|C|}$, where C is the set of testing cases, App_{c_i} is the ground-truth, and Lc_i is the prediction list at the i-th testing case. $I(\cdot)$ is an indicator function which equals 1, when $App_{c_i} \in L_{c_i}$, and equals 0, otherwise. Finally, the average recall is the average of the recall values of all users.

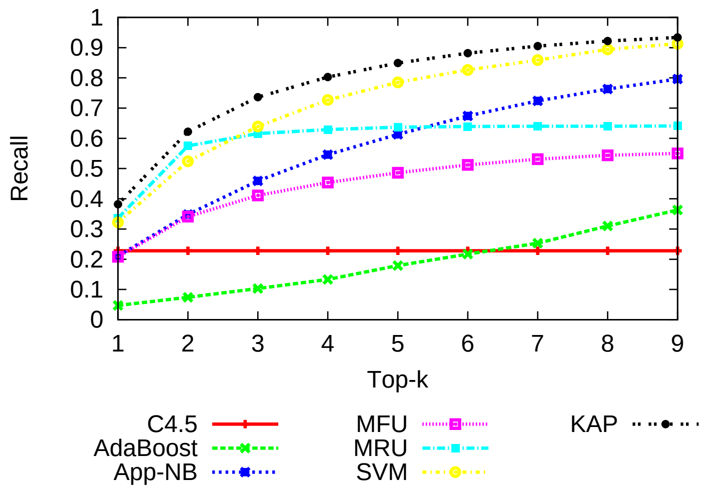
nDCG Score: To evaluate the accuracy of the order of the prediction list, we also test the nDCG score of the prediction results. The IDCG score is fixed to 1 because there is only one used App in the ground-truth. The DCG score is $\frac{1}{\log_2(i+1)}$ when the used App is predicted at position i of the prediction list. Then, nDCG is the average of $\frac{DCG}{IDCG}$ for all testing cases.

6.3 Experimental Results

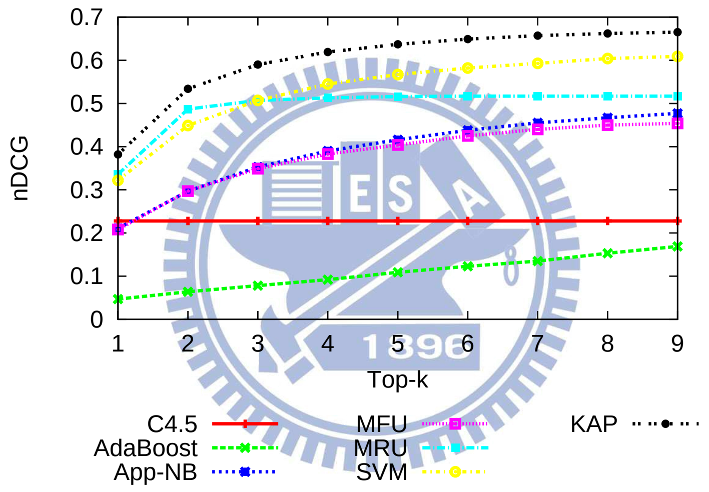
To evaluate the performance of predicting Apps usage by the proposed KAP framework, we first evaluate the overall performance when predicting different numbers of Apps. Then, we test the performance of the personalized feature selection algorithm. The impact of different parameters for the KAP framework and kNN classification is also included. Note that we use top-k = 4, kNN=40%, and the minimum data coverage of personalized feature selection as 70% to be the default parameter settings throughout the experiment.

6.3.1 Overall Performance

First, we evaluate the performance KAP and other different methods under various numbers of prediction, k. As can be seen in Figure 6.1, when the number of prediction k increases, both the recall and nDCG values also increase. However, KAP , App-NB and SVM perform better than others. In Figure 6.1(a), when $k = 9$ (the number of predictions shown in the latest Android system), the recall of KAP and SVM could be more than 90%, while it is below



(a) Recall

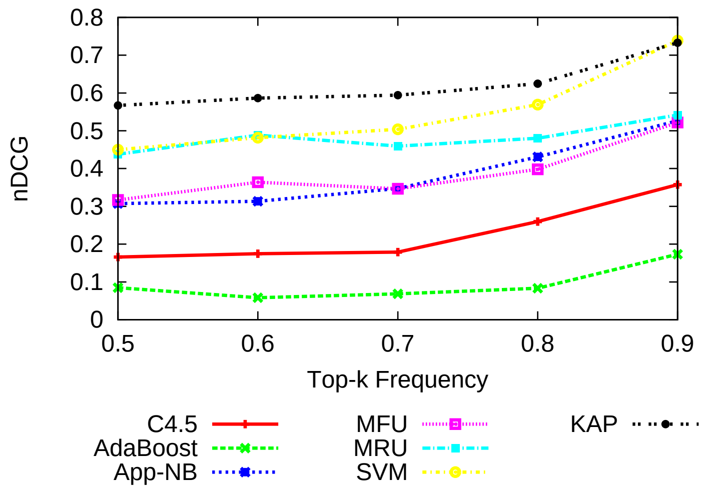


(b) nDCG

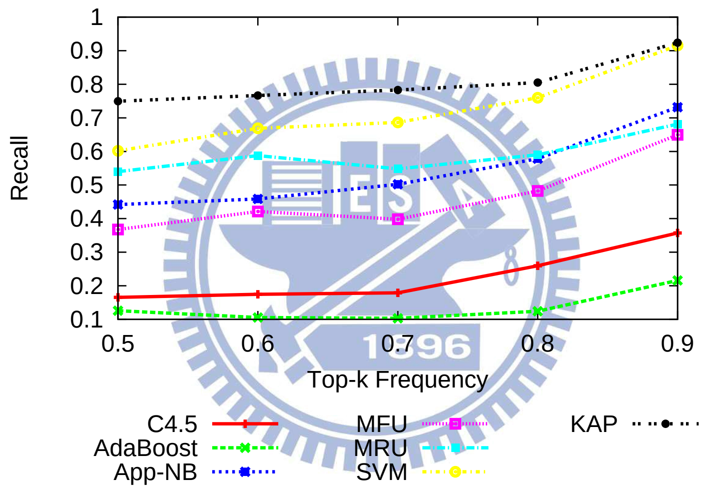
Figure 6.1: Impact of the number of prediction, k.

80% for the others. On the other hand, the nDCG value of KAP shown in Figure 6.1(b) is always higher than that of the other methods, which means the prediction order of KAP is better.

Second, we test the accuracy of varied top-k frequency. The top-k frequency is defined as the ratio of the usage of the most frequent k Apps. For example, if a user has 5 Apps and the usage counts are 3, 1, 2, 5, and 2, the top-2 frequency is thus $\frac{5+3}{3+1+2+5+2} = \frac{8}{13}$. Figure 6.2 shows the results when top-k = 4. Intuitively, when the top-k frequency increases, the accuracy of the MFU method could be better. However, in Figure 6.2(a), even when the ratio is 0.9, the MFU method performs just closed to MRU and App-NB, but worse than both KAP and



(a) Recall



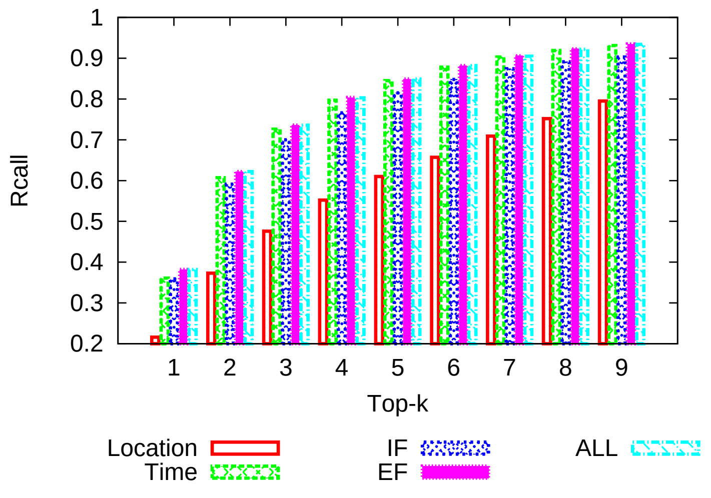
(b) nDCG

Figure 6.2: Impact of top-k frequency.

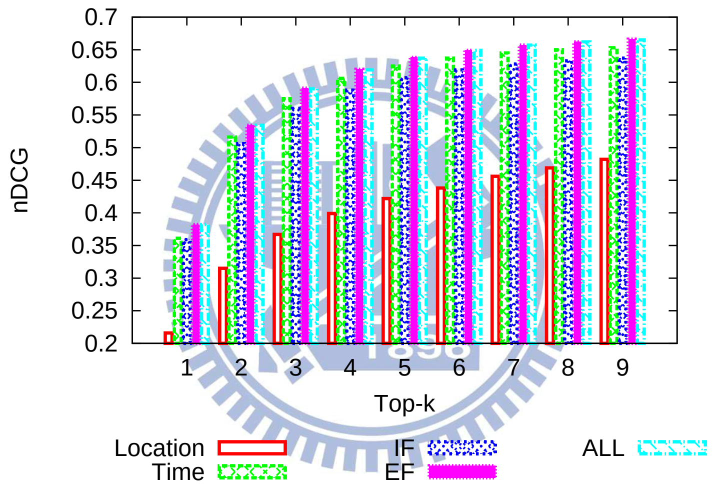
SVM. In Figure 6.2(b), the prediction order of KAP is also better than the results of the other methods.

6.3.2 Impact of Features

Here, we evaluate the performances of using different sets of features for kNN classification. Five kNN-based methods are considered: 1) ALL: use all sensors readings with implicit features, 2) EF: use only the EF feature, 3) IF: use only the IF feature, and 4) Time: use only the temporal features. 5) Location: use only the spatial features. Figure 6.3 shows the results



(a) Recall



(b) nDCG

Figure 6.3: Impact of Features.

of the impact of using different features. Time, IF, EF and ALL have better performances as the top-k candidates are more. The performances are similar when KAP uses certain set of features to predict Apps usage. As a result, the experimental result displays the scalability of KAP that performs well under varied sets of features.

6.3.3 Impact of Personalized Feature Selection

For the proposed KAP method, we evaluate the performance of the personalized feature selection to see if the proposed MDL-based selection algorithm could reduce the used storage

Table 6.1: The storage consumption and accuracy under varied data coverage ρ.

Coverage(%)	30	40	50	60	70	80	90	100
Storage(KB)	28	31	34	37	43	52	82	94
Recall	0.78	0.78	0.80	0.80	0.82	0.82	0.82	0.83
nDCG	0.50	0.51	0.52	0.53	0.55	0.57	0.57	0.58

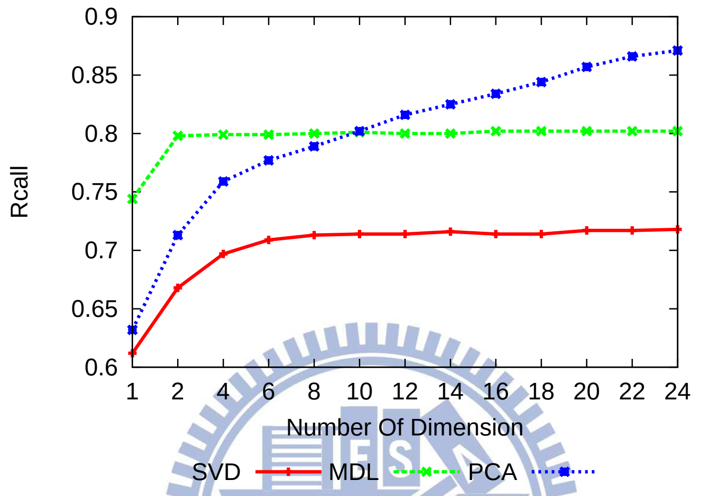
Table 6.2: The execution time of KAP with and without personalized feature selection.

Execution time (ms)	Training	Testing	Total
KAP	86	160	246
KAP without selection	185	160	345

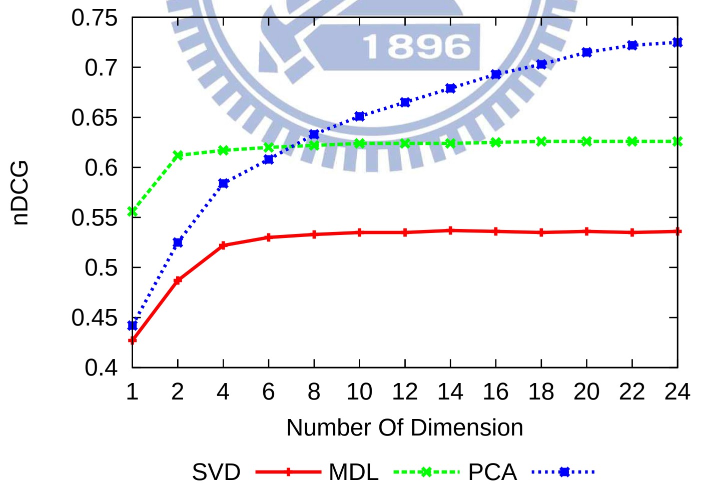
when maintaining a good prediction accuracy. For one user, the average used storage and prediction accuracy is shown in Table 6.1 under different data coverage ρ. As can be seen in Table 6.1 the personalized feature selection could reduce 55% of training data size and only lose 1% of recall and 3% of nDCG when the data coverage is 70%. In addition, Table 6.2 compares the execution time of KAP with and without the personalized feature selection, where the training time is reduced dramatically under $\rho = 70\%$.

6.3.4 Comparison of Different Feature Selection Methods

To exhibit the capability of the proposed MDL-based selection algorithm, we compare the performances with another two mathematical procedures called SVD [1] and PCA [7] in different number of dimensions . In Figure 6.4, the evaluation result of MDL is better than the others when the number of dimension is less than about 6 and turns into stable after 6. The performance of MDL is worse than PCA when the number of dimension is high, but the MDL is the only one that can fit the advantages of storage and power consumption by choosing the useful subset of features. Because SVD and PCA are needed to collect all sensor data in their analysis when the intrinsic feature spaces are transformed into lower feature spaces while preserving as much information as possible.

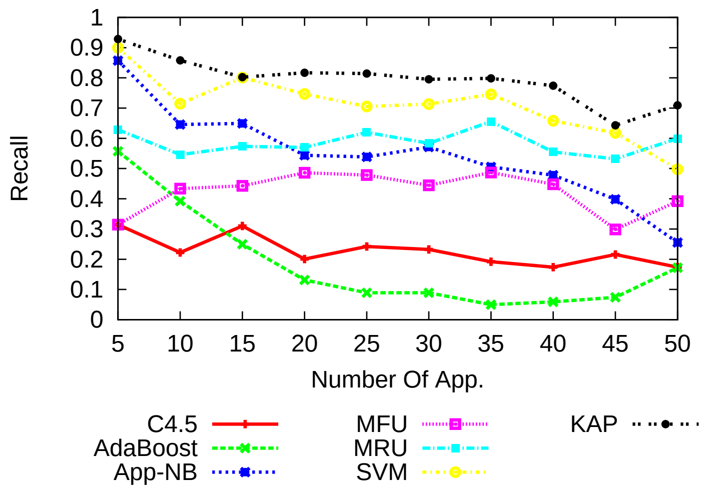


(a) Recall

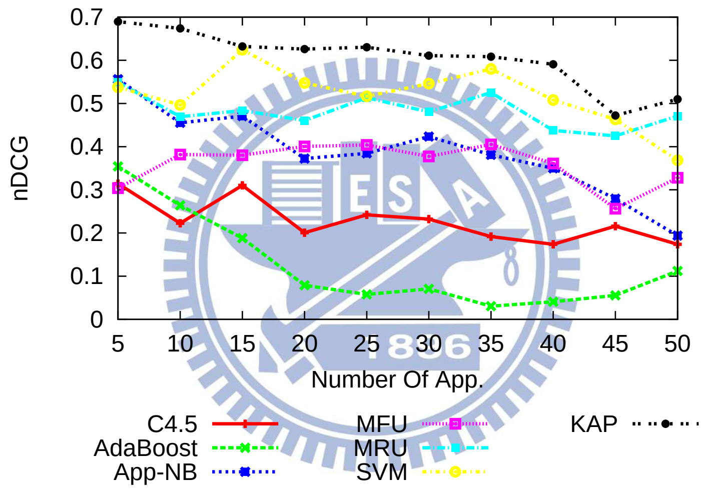


(b) nDCG

Figure 6.4: Comparison of Different Feature Selection Methods



(a) Recall

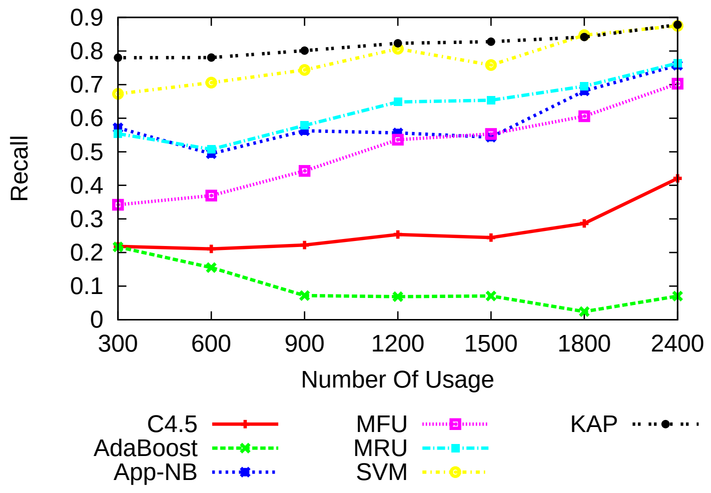


(b) nDCG

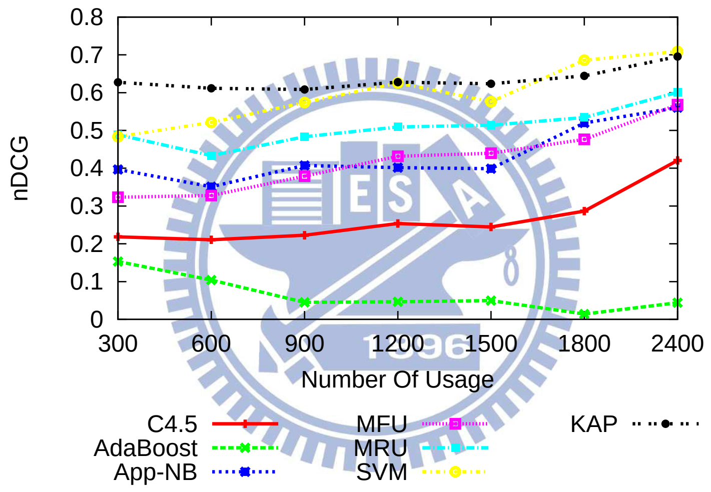
Figure 6.5: Impact of the number of Apps.

6.4 Comparison of Different Usage Behavior

Since different users have different usage behavior, which could extremely affect the prediction accuracy. In this section, we separate users into different groups according to 1) number of installed Apps, 2) usage frequency, and 3) usage entropy. Then, we test the performance of applying different methods on different groups.



(a) Recall

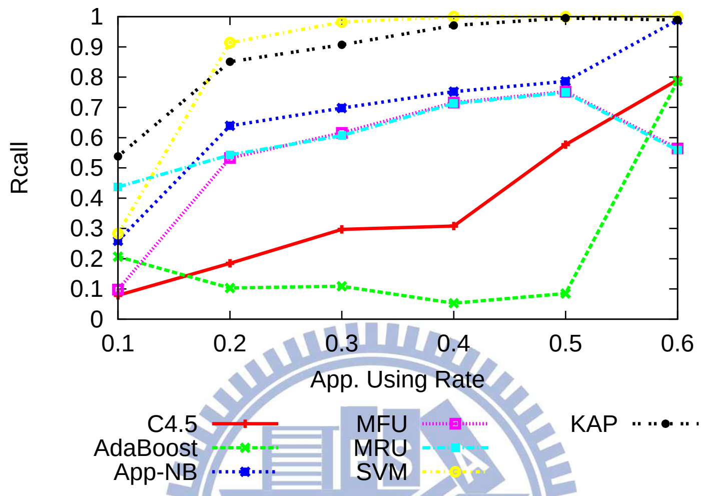


(b) nDCG

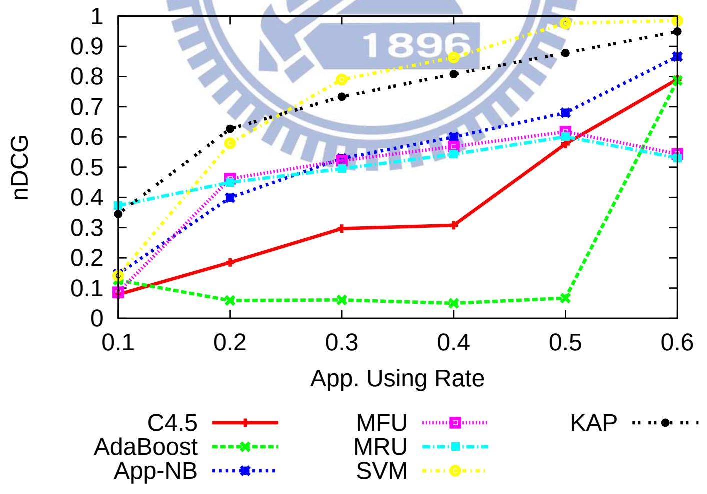
Figure 6.6: Impact of the usage count.

6.4.1 Impact of the Number of Installed Apps

When users launch more Apps, it becomes more difficult to accurately predict Apps usage. Figure 6.5 shows the recall and ndcg results for a varying number of used Apps. As can be seen in Figure 6.5, both the recall and ndcg values decrease when the number of used Apps increases for all methods. However, the decreasing rate of the proposed KAP method is much smoother than that of the others. The recall of KAP is around 85% while that of the others is below 40% when the number of used Apps is 30.



(a) Recall



(b) nDCG

Figure 6.7: Impact of the App using rate.

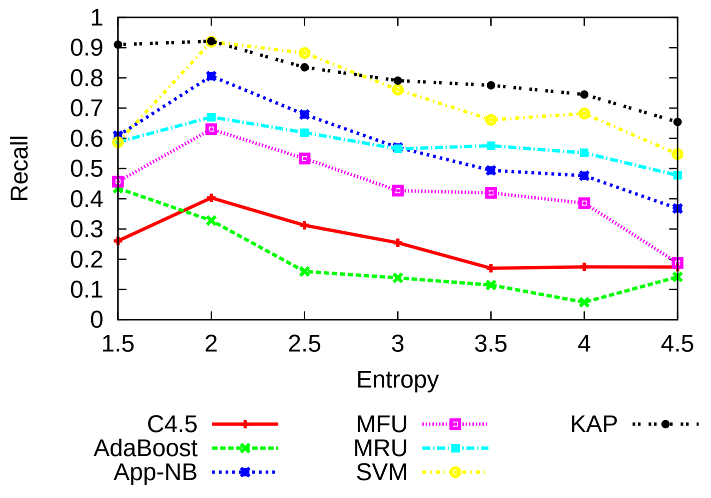
6.4.2 Impact of the Usage Count

Now, we test the impact of the usage count. A higher usage count means we could have more training data to learn the classification model for App prediction. Concurrently, it provides more complicated information of users' usage behavior, and could make noisy data. Figure 6.6 shows the recall and ndcg values. The performance of KAP, Naive Bayes, Decision Tree, and SVM goes up when the usage count increases. However, AdaBoost has worse performance as the usage count goes up. The result shows that the KAP algorithm can handle more complicated and noisy data.

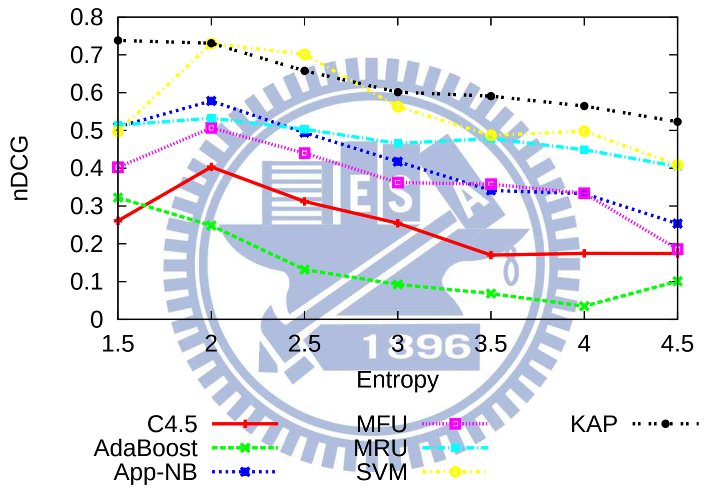
In addition, we also test the impact of the using rate of an App. For example, if a user has 5 Apps and the usage counts are 3, 1, 2, 5, and 2, the App using rates are $\frac{3}{13}$, $\frac{1}{13}$, $\frac{2}{13}$, $\frac{5}{13}$, and $\frac{2}{13}$ respectively. A higher using rate means that the App was often used in the past, which also means the App is very used to the user. On the contrary, a lower using rate means we have less clues to suggest user launching the App in advance. However, those infrequently used Apps are still useful to users such as game, map, productivity, etc. In Figure 6.7, KAP is slightly better than MRU when the App using rate is 0.1, but significantly overwhelming the others. In short, the performances are better as the App using rate is high, while KAP outperforms the others in most cases.

6.4.3 Impact of the Entropy of the Apps Usage

We evaluate the impact of the entropy of the Apps usage. Intuitively, as the entropy of the Apps usage becomes larger, the Apps usage is almost random, and the performance of Apps usage prediction would become worse. Figure 6.8 depicts that the proposed KAP could have around 80% accuracy when the entropy goes to 3 where the other methods except for SVM only have accuracy of less than 50%.



(a) Recall



(b) nDCG

Figure 6.8: Impact of the entropy of Apps.

6.5 Impact of Different Parameters

6.5.1 Number of Iterations for Implicit Feature Extraction

First, we test the number of iterations of deriving the implicit feature for each testing case. As shown in Table 6.3, the accuracy stays almost the same after the second iteration. This indicates that the iterative refinement algorithm could converge within 2 iteration which is sufficient to estimate the implicit feature.

Table 6.3: The recall and nDCG values under varied numbers of iterations.

#Iterations	1	2	3	4	5
Recall	0.67	0.79	0.79	0.80	0.81
nDCG	0.43	0.59	0.59	0.60	0.61

Table 6.4: The recall and nDCG values under varied minimum probability for session identification.

min_{tp}	0.5	0.25	0.1	0.075	0.05	0.025	0.001
Recall	0.73	0.77	0.83	0.81	0.80	0.75	0.74
nDCG	0.53	0.57	0.61	0.58	0.55	0.53	0.52

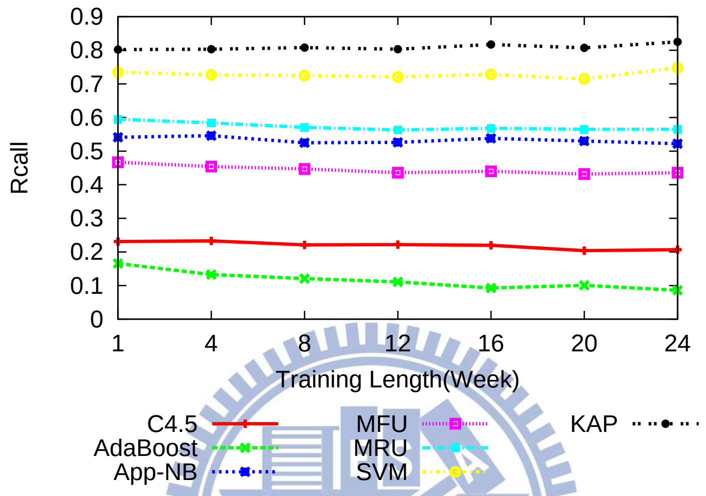
6.5.2 Minimum Probability for Identifying Usage Sessions

As users usage sessions could be varied according to different tasks, we only need the useful length of the usage sessions to perform accurate Apps usage prediction, instead of calculate the full usage sessions. Therefore, we conduct this experiment to evaluate the impact of the length of usage sessions. Ac can be seen in Table 6.4, the results are not affected by the minimum transition probability, min_{tp}, too much. From our collected data, the session length is around 2 when min_{tp} is 0.5, and the best case is under $min_{tp} = 0.1$, which has the session length as around 5.

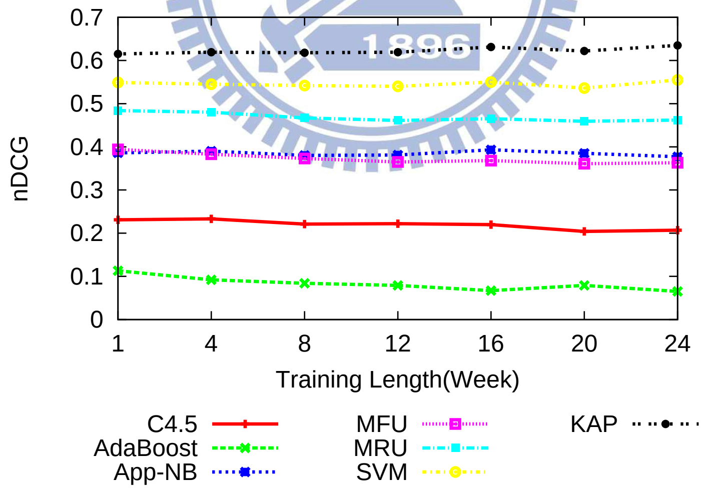
6.5.3 Parameters for kNN Classification

There are two main parameters affecting the accuracy of kNN classification: 1) the length of the training period, and 2) the number of neighbors of kNN. Here, we fix the number of predictions to 4 Apps and compare the recall and nDCG values of KAP and the other methods. Figures 6.9(a) and 6.9(b) show the results, where the recall and nDCG values of the KAP, MFU, MRU, and SVM methods almost keep the same performance under varied training lengths. Therefore, we suggest that we just need to collect a short period as training data to predict users' Apps usage, since users' behavior is considered as stable over a short period.

Then, we evaluate the impact of selecting different numbers of neighbors to perform kNN classification. Because the training data of different users could vary from several hundreds



(a) Recall



(b) nDCG

Figure 6.9: Impact of training length.

Table 6.5: The recall and nDCG values under varied number of neighbors for kNN.

kNN(%)	20	40	60	80	100
Recall	0.74	0.79	0.80	0.80	0.81
nDCG	0.55	0.61	0.63	0.63	0.64

Table 6.6: The recall and nDCG values under varied λ for recency factor.

λ(hour)	1	12	24	48	72	96	120
Recall	0.80	0.78	0.76	0.73	0.72	0.71	0.71
nDCG	0.62	0.58	0.56	0.54	0.53	0.53	0.52

to thousands. we use a relative value for the number of neighbors. Table 6.5 shows the results of the recall and nDCG values for different number of neighbors. As can be seen in Table 6.5, even only select 40% of training data as the neighbors, the recall value is almost 80%. Therefore, we set the default number of neighbor as 40% throughout the experiments.

6.5.4 Parameters for kNN Voting Function

Finally, we test the impact of the decaying parameter λ in the voting function expressed in Equation(5.1). The λ affects the importance of Apps according to their last used time. Namely, every interaction decays exponentiation over time with a half life λ. As shown in Figure 6.6, the best value occurs at 1 hour, and then the performances are worse when the λ is higher.

Chapter 7

Conclusion

In this chapter, we propose an Apps usage prediction framework, KAP, which predicts Apps usage regarding both the explicit readings of mobile sensors and the implicit transition relation among Apps. For the explicit feature, we consider three different types of mobile sensors: 1) device sensors, 2) environmental sensors, and 3) personal sensors. For the implicit features, we construct an Apps Usage Graph (AUG) to model the transition probability among Apps. Then, for each training datum, we could represent the next used App as the implicit feature which describes the probability of transition from other Apps. Note that, since the next App in the testing data is unknown, we propose an iterative refinement algorithm to estimate both the probability of the App to be invoked next and its implicit feature. We claim that different usage behaviors are correlated to different types of features. Therefore, a personalized feature selection algorithm is proposed, where for each user, only the most relative features are selected. Through the feature selection, we can reduce the dimensionality of the feature space and the energy/storage consumption.

We integrate the explicit and implicit features as the feature space and the next used App as the class label to perform kNN classification. In the experimental results, our method outperforms the state-or-the-art methods and the currently used methods in most mobile devices. In addition, the proposed personalized feature selection algorithm could maintain better performance than using all features. We also evaluate the performance of KAP for different types of users, and the results show that KAP is both adaptive and flexible.

Bibliography

[1] Peter A. Businger and Gene H. Golub. Algorithm 358: singular value decomposition of a complex matrix [f1, 4, 5]. *Commun. ACM*, 12(10):564–565, 1969.

[2] Shuo Chen, Joshua L. Moore, Douglas Turnbull, and Thorsten Joachims. Playlist prediction via metric embedding. In *The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '12, Beijing, China, August 12-16, 2012*, pages 714–722, 2012.

[3] Driss Choujaa and Naranker Dulay. Predicting human behaviour from selected mobile phone data points. In *UbiComp 2010: Ubiquitous Computing, 12th International Conference, UbiComp 2010, Copenhagen, Denmark, September 26-29, 2010, Proceedings*, pages 105–108, 2010.

[4] Trinh Minh Tri Do, Jan Blom, and Daniel Gatica-Perez. Smartphone usage in the wild: a large-scale analysis of applications and context. In *Proceedings of the 13th International Conference on Multimodal Interfaces, ICMI 2011, Alicante, Spain, November 14-18, 2011*, pages 353–360, 2011.

[5] Yuxiao Dong, Jie Tang, Sen Wu, Jilei Tian, Nitesh V. Chawla, Jinghai Rao, and Huanhuan Cao. Link prediction and recommendation across heterogeneous social networks. In *12th IEEE International Conference on Data Mining, ICDM 2012, Brussels, Belgium, December 10-13, 2012*, pages 181–190, 2012.

[6] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, 2002.

[7] Ian Jolliffe. *Principal component analysis.* Wiley Online Library, 2005.

[8] Daisuke Kamisaka, Shigeki Muramatsu, Hiroyuki Yokoyama, and Takeshi Iwamoto. Operation prediction for context-aware user interfaces of mobile phones. In *2009 Ninth Annual International Symposium on Applications and the Internet*, pages 16–22, 2009.

[9] Ashraf M Kibriya and Eibe Frank. An empirical comparison of exact nearest neighbour algorithms. In *Knowledge Discovery in Databases: PKDD 2007*, pages 140–151. Springer, 2007.

[10] Ashraf Masood Kibriya. *Fast algorithms for nearest neighbour search.* PhD thesis, The University of Waikato, 2007.

[11] Jacob Kogan. Feature selection over distributed data streams through convex optimization. In *Proceedings of the Twelfth SIAM International Conference on Data Mining, Anaheim, California, USA, April 26-28, 2012*, pages 475–484, 2012.

[12] Neal Lathia, Stephen Hailes, Licia Capra, and Xavier Amatriain. Temporal diversity in recommender systems. In *Proceeding of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2010, Geneva, Switzerland, July 19-23, 2010*, pages 210–217, 2010.

[13] Po-Ruey Lei, Tsu-Jou Shen, Wen-Chih Peng, and Ing-Jiunn Su. Exploring spatial-temporal trajectory model for location prediction. In *12th IEEE International Conference on Mobile Data Management, MDM 2011, Luleå, Sweden, June 6-9, 2011, Volume 1*, pages 58–67, 2011.

[14] Zhung-Xun Liao, Po-Ruey Lei, Tsu-Jou Shen, Shou-Chung Li, and Wen-Chih Peng. Mining temporal profiles of mobile applications for usage prediction. In *12th IEEE International Conference on Data Mining Workshops, ICDM Workshops, Brussels, Belgium, December 10, 2012*, pages 890–893, 2012.

[15] Zhung-Xun Liao, Wen-Chih Peng, and Philip S. Yu. Mining usage traces of mobile applications for dynamic preference prediction. In *17th Pacific-Asia Conference on Knowledge Discovery and Data Mining, PAKDD 2013, Gold Coast, Australia, April 13-17, 2013*, 2013.

[16] David Liben-Nowell and Jon M. Kleinberg. The link prediction problem for social networks. In *Proceedings of the 2003 ACM CIKM International Conference on Information and Knowledge Management, New Orleans, Louisiana, USA, November 2-8, 2003*, pages 556–559, 2003.

[17] Eric Hsueh-Chan Lu, Wang-Chien Lee, and Vincent Shin-Mu Tseng. A framework for personal mobile commerce pattern mining and prediction. *IEEE Trans. Knowl. Data Eng.*, 24(5):769–782, 2012.

[18] M. Matsumoto, R. Kiyohara, H. Fukui, and M. Numao. Proposition of the context-aware interface for cellular phone operations. In *5th International Conference on Networked Sensing Systems, June 17-19, 2008*, pages 233–233, 2008.

[19] Anna Monreale, Fabio Pinelli, Roberto Trasarti, and Fosca Giannotti. Wherenext: a location predictor on trajectory pattern mining. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009*, pages 637–646, 2009.

[20] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

[21] J. Rissanen. Hypothesis selection and testing by the mdl principle. *The Computer Journal*, 42:260–269, 1999.

[22] Salvatore Scellato, Mirco Musolesi, Cecilia Mascolo, Vito Latora, and Andrew T. Campbell. Nextplace: A spatio-temporal prediction framework for pervasive systems. In *Pervasive Computing - 9th International Conference, Pervasive 2011, San Francisco, CA, USA, June 12-15, 2011. Proceedings*, pages 152–169, 2011.

[23] Kent Shi and Kamal Ali. Getjar mobile application recommendations with very sparse datasets. In *The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '12, Beijing, China, August 12-16, 2012*, pages 204–212, 2012.

[24] Choonsung Shin, Jin-Hyuk Hong, and Anind K. Dey. Understanding and prediction of mobile application usage for smart phones. In *The 2012 ACM Conference on Ubiquitous Computing, Ubicomp '12, Pittsburgh, PA, USA, September 5-8, 2012*, pages 173–182, 2012.

[25] Xing Wu, Geoffrey Holmes, and Bernhard Pfahringer. Mining arbitrarily large datasets using heuristic k-nearest neighbour search. In *AI 2008: Advances in Artificial Intelligence*, pages 355–361. Springer, 2008.

[26] Bo Yan and Guanling Chen. Appjoy: personalized mobile application discovery. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services (MobiSys 2011), Bethesda, MD, USA, June 28 - July 01, 2011*, pages 113–126, 2011.

[27] Tingxin Yan, David Chu, Deepak Ganesan, Aman Kansal, and Jie Liu. Fast app launching for mobile devices using predictive user context. In *The 10th International Conference on Mobile Systems, Applications, and Services, MobiSys'12, Ambleside, United Kingdom - June 25 - 29, 2012*, pages 113–126, 2012.

[28] Peifeng Yin, Ping Luo, Wang-Chien Lee, and Min Wang. App recommendation: a contest between satisfaction and temptation. In *Sixth ACM International Conference on Web Search and Data Mining, WSDM 2013, Rome, Italy, February 4-8, 2013*, pages 395–404, 2013.